# Git clone

Je maakt een clone van de remote repository.

git clone ssh://server/path/project.git

Zal een map "project" aanmaken in de map waar je nu in zit en de master branch uitchecken. Je kunt vanaf dan werken met Git.

# Git init en git remote

git init mijnproject

Dit zal een map "mijn project" aanmaken vanuit de huidige map waar je in zit en een lege repository aanmaken.

git remote add origin ssh://server/path/mijnproject.git

Wanneer je thuis bent en wel over internet beschikt "hang" je nieuwe repository aan een remote.

Wanneer je de remote aan je lokale repository hebt gekoppeld en je commits wilt "pushen" doe je de eerste keer:

git push origin master[/b]  
  
Zodat remote de branch wordt aangemaakt. In het vervolg kun je gewoon "git push" doen.  
of git init lekker committen etc en een remote toevoegen.

Git reset

Met het commando git reset kun je eenvoudig bepaalde commits terug draaien.

Stel je hebt twee commits gedaan en nog niet gepushed (belangrijk!) en je hebt toch besloten deze terug te draaien en alles opnieuw te committen,

git reset --soft HEAD^2

# Git revert

Wanneer je een normale commit wilt terug draaien is dit vrij eenvoudig,

Waarbij de hash uiteraard maar een voorbeeld is.

git revert 9c9bc66967ff6a03f40a4627db7231a9cbd9221c

# Git branch

**Lokaal of remote**  
Met branches werk je altijd lokaal. Er is in Git (zover ik weet) geen mogelijkheid om direct op de remote branch te werken. Wanneer je dus een branch aanmaakt staat deze alleen nog lokaal. Je moet de branch pushen om hem ook remote beschikbaar te maken en je code te kunnen delen met andere ontwikkelaars.

git branch swiftmailer

git push origin swiftmailer

Wanneer je wilt weten welke branches er remote allemaal zijn, welke je nu uitgechecked hebt of welke branches bij je lokaal staan voer je het volgende commando uit:  
  
branch –a

Dit commando geeft je een volledige lijst van alle branches zowel lokaal als remote, en de branch met het sterretje ervoor is de branch waar je momenteel op zit. Een voorbeeld:

git branch -a  
  swiftmailer  
\* master  
  remotes/origin/swiftmailer  
  remotes/origin/master

git push origin master

Het gevolg was dat ik mijn branch ging mergen in de master terwijl deze nog niet klaar was. Omdat dit met 'push' meteen remote staat moet je met "git revert --mainline" gaan spelen om het terug te draaien. Wanneer je een merge conflict hebt, dan heb je geluk en kun je nog "git reset --soft ...." doen om ervoor te zorgen dat de merge terug gedraait wordt. Het is daarom een wijze les om alleen bij het aanmaken van de branch de eerste keer "origin swiftmailer" erachter te zetten **in de swiftmailer branch** en in het vervolg alleen "git push" te doen.

# Git checkout

Met "git checkout" kun je dus schakelen naar een andere branch om daar verder in te werken. Wanneer je dit doet zal git alle bestanden inchecken die in die branch zitten.

Belangrijk is dat alle wijzingen die je nog niet gestaged hebt (met "git add") meegenomen worden als unstaged files naar de branch waarin je uitchecked. Wanneer ze al wel gestaged waren krijg je een melding dat je ze moet stashen of committen. Stashen betekend parkeren.

git checkout swiftmailer

**Uitchecken naar een branch die nog niet lokaal staat?**  
Stel een collega ontwikkelaar heeft een nieuwe branch gemaakt en deze remote beschikbaar gemaakt. Wanneer hij of zij dat gedaan heeft betekend dat nog niet dat jij automatisch beschikking hebt over deze branch. Om aan deze branch verder te werken zul je eerst de branch lokaal moeten maken, uitchecken en een pull doen om alle remote content binnen te halen. Je zou dit als volgt kunnen doen.

git branch naambranch  
git checkout naambranch  
git pull origin naambranch

Maar met de -b optie is er ook een verkorte versie:

git checkout -b naambranch  
git pull origin naambranch

Op die manier heb je dus de branch aangemaakt en meteen uitgechecked.

# Git pull

Met "git pull" kun je de remote commits die nog niet bij jou lokaal staan ophalen. Je hebt ook "git fetch" maar deze gebruik ik zelf niet vaak, maar zal ik later wellicht nog toevoegen aan deze tutorial.

**Faster-forward**  
Git pull probeert standaard een faster-forward uit te voeren op je lokale branch. Dit beteken dat Git kijkt of je eigen branch niet voorloopt op de remote en als dat niet het geval is dan kunnen de remote commits op je HEAD gezet worden zonder conflicten en is de pull klaar.

**Git pull repository:branch**  
Wanneer je een "git pull" doet zonder specificatie (wordt ook refspec genoemd) dan zal Git proberen de remote commits op te halen in de branch waar je op zit. Wanneer je lokale branch naam anders is als remote moet je dit wel goed configureren anders zal git je vertellen dat hij niet weet waar hij de content vandaan moet halen. Maar dit zal niet vaak voorkomen. Je kunt in zo'n geval ook zelf aangeven waar Git de content uit moet halen:

git pull origin master

Hier kleeft alleen een gevaar aan, dat is dat wanneer je doet vanuit de verkeerde branch ("anderebranch" bijvoorbeeld) dat Git zal proberen deze branches te mergen. Als je hier op tijd achterkomt is dat niet zo'n probleem dan doe je:

git reset --hard HEAD^1

Dat wilt zeggen, draai 1 commit terug. Die ene commit is dus je merge commit. Als je nog in merge status bent gebruik je logischerwijs HEAD ipv HEAD^1. Wanneer je merge al gepushed hebt wordt het lastiger, dan is "git revert" je redding.

**Verder over Git pull..**  
Feitelijk (dat is ook wat de help zegt) is Git pull niets anders als een git fetch met een git merge erachter aan, en dat alle opties die git pull kent git fetch of git merge ook kent.

# Git merge en git rebase

Een andere prachtige feature van git is 'git rebase'.  
  
Stel je wilt alle wijzigingen van 'anderebranch' in de master zetten. Dan kun je voor een merge kiezen, maar soms is rebasen eenvoudiger. Dit is vooral wanneer je de commits die op 'anderebranch' zijn gedaan overzichtelijk terug wilt kunnen zien na de rebase. Het is dan ook eenvoudiger om een individuele commit uit 'anderebranch' in de master later terug te draaien met "git revert". Ook blijft je tijdlijn mooier doordat je niet weer een aftakking maakt van je master branch en deze weer samenvoegt. Als je het programma "gitk" geïnstalleerd hebt zal het je snel duidelijk worden wat ik daarmee bedoel.  
  
Meestal doe ik een rebase als ik weet dat 'anderebranch' maar 5 commits heeft bijvoorbeeld. Om een hele merge te maken vindt ik dan overbodig. Wanneer je een rebase doet zal Git proberen alle commits die in 'anderebranch' zijn gedaan in de juiste tijdsvolgorde 'opnieuw af te spelen'. Het verschil met een merge is dat die bovenop je HEAD wordt gezet. Een andere branch rebasen gaat als volgt:

git rebase anderebranch

Als je dan in de master uitgechecked bent zal Git proberen alle commits in 'anderebranch' in de juiste tijdsvolgorde in te voegen in de master. Nog een voordeel t.o.v. van een merge is dat je in je Git log geen meldingen ziet die lijken op: "Merge branch 'anderebranch' of <ssh://host/var...> master" die weinig zeggen. Je kunt deze melding bij een merge wel overschrijven/wijzigen door:

git merge anderebranch -m "Cool new feature"

**Rebase via pull**  
Wanneer je "git pull" doet zal Git automatisch een merge doen als er geen faster-forward gedaan kan worden. Feitelijk worden jouw commits die niet remote staan, en remote commits die niet bij jou lokaal staan samengevoegd in een merge commit. Ik vindt dit vaak niet wenselijk omdat anderen dan niet goed kunnen zien wat je nou precies aangepast hebt. Bij een rebase zal git proberen jouw commits proberen in te voegen in de juiste tijdsvolgorde tussen de andere remote commits. Zo is overzichtelijk wie wat gedaan heeft.  
  
Je doet dit als volgt:

git pull --rebase[/b]  
  
En het vervolg process zal precies hetzelfde zijn als hierboven. Wanneer een faster-forward mogelijk is zal Git de rebase negeren (want je loopt tenslotte niet voor op de remote branch). Het oplossen van conflicten gaat precies hetzelfde!  
  
Met merge en rebase kan nog veel meer. Bijvoorbeeld interactief rebasen. En bij "git merge" en "git rebase" kun je nog een tal van opties meegeven. Ik kan je zeker aanraden om eens de help goed door te lezen van beiden en met het e.e.a. eens te experimenteren.